PAPER *Special Section on VLSI Design and CAD Algorithms*

Design Method for Numerical Function Generators Using Recursive Segmentation and EVBDDs[*]

Shinobu NAGAYAMA[†a)], Tsutomu SASAO[††b)], *and* Jon T. BUTLER[†††c)], *Regular Members*

SUMMARY Numerical function generators (NFGs) realize arithmetic functions, such as e^x, $\sin(\pi x)$, and $\sqrt{x}$, in hardware. They are used in applications where high-speed is essential, such as in digital signal or graphics applications. We introduce the edge-valued binary decision diagram (EVBDD) as a means of reducing the delay and memory requirements in NFGs. We also introduce a recursive segmentation algorithm, which divides the domain of the function to be realized into segments, where the given function is realized as a polynomial. This design reduces the size of the multiplier needed and thus reduces delay. It is also shown that an adder can be replaced by a set of 2-input AND gates, further reducing delay. We compare our results to NFGs designed with multi-terminal BDDs (MTBDDs). We show that EVBDDs yield a design that has, on the average, only 39% of the memory and 58% of the delay of NFGs designed using MTBDDs.

key words: *Edge-valued binary decision diagrams (EVBDDs), recursive segmentation, piecewise polynomial approximation, numerical function generators (NFGs), programmable architecture.*

1. Introduction

The computation of arithmetic functions, such as trigonometric, logarithmic, square root, and reciprocal functions, has a long history. More than 150 years ago, Charles Babbage designed the difference machine to compute polynomials that could be used to approximate other functions, such as logarithms [27]. The introduction of electronic general purpose computers and special languages like FORTRAN improved upon the speed and ease at which arithmetic functions could be calculated. Well into the beginning of the 21st century, the goals remain the same – to compute functions at high-speed and with relative ease by the user.

In this paper, we propose a design of a programmable numerical function generator (NFG) that computes an arithmetic function in fixed-point representation. It takes advantage of large quantities of inexpensive, programmable logic available in modern FPGAs. Because of FPGAs, there has been recent interest in NFGs that realize polynomials that approximate the given function [4, 6–8, 18, 25, 26]. In the past, designs have used *uniform* segments across the function's domain, where, in each segment, a (generally different) polynomial is used to realize the function. By decreasing the segment size, any desired accuracy can be achieved. Accuracy also depends on the order of the polynomial used in the approximation. Linear [26] and higher order approximations have been considered [4, 6–8, 18, 25]. Linear approximation and uniform segmentation are well suited for some functions like 2^x, $\sin(\pi x)$, and $\cos(\pi x)$, but are not appropriate for other functions like $\sqrt{-\ln(x)}$ and the entropy function $-(x\log_2(x)+(1-x)\log_2(1-x))$. For such functions, *non-uniform* segmentation produces realizations with the desired accuracy [3]. In this case, segments are chosen to be as wide as possible while still achieving the specified accuracy. As a result, the segment width is adapted to the local characteristics of the function – wide segments where the function is nearly linear and narrow segments where the function is nonlinear. It follows that this yields the fewest segments needed to achieve the given accuracy. Since the coefficients of the approximating polynomial are stored in local memory, non-uniform segmentation offers a way to reduce the memory requirements of an NFG realized by a memory-constrained FPGA.

In this paper, we propose a new segmentation algorithm and a new programmable architecture. Specifically, we propose the edge-valued binary decision diagram (EVBDD) as a way to design a programmable circuit that maps a given X into a segment, where the function $f(X)$ is realized by a specific polynomial. We also propose a recursive segmentation algorithm that produces segments whose widths are chosen especially to simplify the hardware, while adapting to the degree of nonlinearity of $f(X)$. This approach is a hybrid of an approach [10, 11] that uses a *special (non-optimum) non-uniform* segmentation and another [22, 24] that uses the *optimum* non-uniform segmentation.

This paper is divided as follows. The next section introduces preliminary concepts, including an introduction to the EVBDD. In Section 3, we discuss the segmentation of the domain, including a new recursive segmentation method. In Section 4, we discuss the architecture of our proposed NFG. Experimental results are shown for the realization of various functions in Section 5. Here, we compare our results to another programmable architecture that is designed using the multi-terminal BDD (MTBDD). Finally, in Section 6, we

Manuscript received March 6, 2007.
Manuscript revised June 18, 2007.
Final manuscript received August 1, 2007.

[†]The author is with the Department of Computer Engineering, Hiroshima City University, Hiroshima-shi, 731–3194 Japan.
[††]The author is with the Department of Computer Science and Electronics, Kyushu Institute of Technology, Iizuka-shi, 820–8502 Japan.
[†††]The author is with the Department of Electrical and Computer Engineering, Naval Postgraduate School, Monterey, CA 93943-5121 USA.
a) E-mail: nagayama@ieee.org
b) E-mail: sasao@cse.kyutech.ac.jp
c) E-mail: jon_butler@msn.com

[*]This paper is an extension of [16].

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **JUN 2007**	2. REPORT TYPE	3. DATES COVERED
4. TITLE AND SUBTITLE **Design Method for Numerical Function Generators Using Recursive Segmentation and EVBDDs**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Postgraduate School,Department of Electrical and Computer Engineering,Monterey,CA,93943**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**10**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

provide concluding remarks.

2. Preliminary Concepts

2.1 Number Representation and Error

The value of the input variable X and the function value $f(X)$ are represented in fixed point. Specifically,

Definition 1: X has a **fixed-point representation** $X = (x_{l-1}\ x_{l-2}\ \ldots x_1\ x_0.\ x_{-1}\ x_{-2}\ \ldots\ x_{-m})_2$, where $x_i \in \{0,1\}$, l is the number of bits in the integer part, and m is the number of bits in the fractional part. Each bit x_i contributes $2^i x_i$ to the **value** of X except, x_{l-1}, which contributes $-2^{l-1}x_{l-1}$. That is, the fixed-point representation is in 2's complement.

Definition 2: **Error** is the absolute difference between the exact value and the value produced by the hardware. **Acceptable error** is the maximum error that an NFG may assume; it is usually a specification to be satisfied by the hardware. **Approximation error** is the error caused by a function approximation. **Acceptable approximation error** is the maximum approximation error that a function approximation may assume. **Rounding error** is the error caused by removing certain least significant bits either by rounding or by truncation.

Definition 3: **Precision** is the total number of bits for a binary fixed-point representation. Specifically, n**-bit precision** specifies that n bits are used to represent the number; that is, $n = l + m$. We assume that an n**-bit precision NFG** has an n-bit input.

Definition 4: **Accuracy** is the number of bits in the fractional part of a binary fixed-point representation. m**-bit accuracy** specifies that m bits are used to represent the fractional part of the number. When the maximum error is 2^{-m}, the accuracy can be expressed as 1 **unit in the last place (ULP)**. In this paper, an m**-bit accuracy NFG** is an NFG with an m-bit fractional part of the input, an m-bit fractional part of the output, and a 1 ULP error.

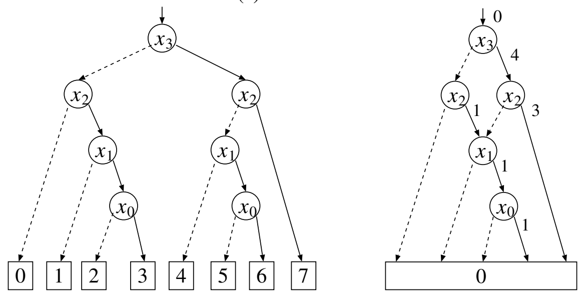
2.2 Edge-Valued Binary Decision Diagram

Definition 5: A binary decision diagram (BDD) [2] is a rooted directed acyclic graph representing a logic function: $\{0,1\}^n \to \{0,1\}$. The BDD is obtained by repeatedly applying the Shannon expansion to the logic function. Each function, including the original function and all sub-functions resulting from applying the Shannon expansion, is represented by a non-terminal node, unless that function is a trivial function, 0 or 1, in which case, it is represented by a terminal node. A non-terminal node has two outgoing edges, a 0-edge and a 1-edge, that correspond to the values of input variables. A terminal node has no outgoing edges.

Definition 6: A **multi-terminal BDD (MTBDD)** [5,19] is an extension of the BDD, and represents an integer function: $\{0,1\}^n \to Z$, where Z is a finite set of integers. Specifically,

x_3	x_2	x_1	x_0	f	x_3	x_2	x_1	x_0	f
0	0	0	0	0	1	0	0	0	4
0	0	0	1	0	1	0	0	1	4
0	0	1	0	0	1	0	1	0	5
0	0	1	1	0	1	0	1	1	6
0	1	0	0	1	1	1	0	0	7
0	1	0	1	1	1	1	0	1	7
0	1	1	0	2	1	1	1	0	7
0	1	1	1	3	1	1	1	1	7

(a) Function table.



(b) MTBDD. (c) EVBDD.

Fig. 1 MTBDD and EVBDD for an integer function.

it is a BDD in which the terminal nodes are not restricted to 0 and 1. Rather, they are labeled by integer values.

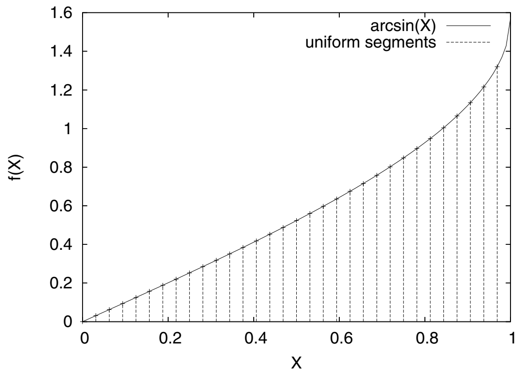
Definition 7: An **edge-valued BDD (EVBDD)** [9,19] is an extension of the BDD, and represents an integer function. An EVBDD consists of one terminal node representing 0 and non-terminal nodes with a weighted 1-edge, where the weight is an integer. Note that, in the EVBDD, 0-edges have weight 0.

Example 1: Fig. 1(b) and (c) show an MTBDD and an EVBDD for the integer function f defined by Fig. 1(a). In Fig. 1(b) and (c), dashed lines and solid lines denote 0-edges and 1-edges, respectively. Note that the EVBDD has weighted 1-edges. In the MTBDD, terminal nodes represent function values. Thus, to evaluate the function, we traverse the MTBDD from the root node to a terminal node according to the input values, and obtain the function value (an integer) from the terminal node. On the other hand, in the EVBDD, we obtain the function value by summing the weights of the edges traversed from the root node to the terminal node. (End of Example)

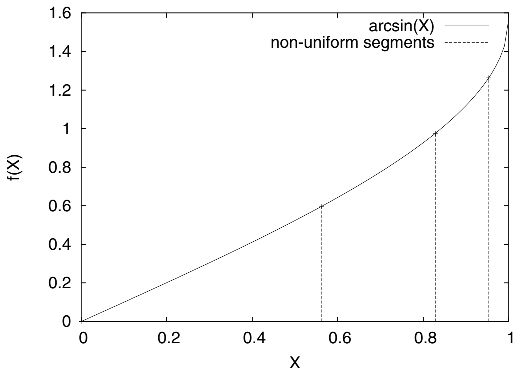
3. Piecewise Polynomial Approximation Based on Non-uniform Segmentation

3.1 Uniform and Non-uniform Segmentations

The realization of a function $f(X)$ in hardware is done by dividing the domain X of the function into *segments*. We choose non-uniform segments, which means that each segment width is chosen so that the given acceptable approximation error ε_a is just met. Therefore, if the function is close to linear in a linear approximation, then a wide segment occurs. And, if the function is highly nonlinear, a narrow segment occurs. In each case, the maximum error in



(a) Uniform segmentation.



(b) Non-uniform segmentation.

Fig. 2 Uniform and non-uniform segmentations of $\arcsin(X)$.

the segment is ε_a. Finding an optimum segmentation is an important part of the design process. Within each segment, a polynomial is used to approximate $f(X)$ in that segment. If the segment is sufficiently small, the polynomial will approximate $f(X)$ to the desired accuracy. For example, if the segment is very small, even a linear approximation will be sufficiently accurate. However, when the segment size is small, too many segments may be required, and there may not be enough memory to store all the required coefficients. Thus, for memory-constrained implementations (e.g. FPGA), it is important to reduce the number of segments while achieving the desired accuracy. There are two methods to reduce the number of segments.

One uses a higher order polynomial to approximate the function. In general, a higher order polynomial results in larger segments, and so reduces the number of segments. However, for certain functions like $\sqrt{-\ln(X)}$ and the entropy function, just using a higher order polynomial cannot reduce the number of segments effectively. Most of existing methods [4,6–8,18,25,26] use *uniform segmentation*, which partitions the domain into segments with the same size. In such a segmentation, the most significant bits of X are used to specify a segment, and the least significant bits determine a point within that segment. The size of all segments is the same as the smallest segment size needed to achieve the desired accuracy. Therefore, depending on functions, uniform segmentation can yield too many segments even if a higher order polynomial is used [15,17].

To reduce the number of segments for such functions, there is another method, *non-uniform segmentation*. In this method, segments are chosen to be as wide as possible while still achieving the desired accuracy. Such an optimum non-uniform segmentation yields the fewest segments for the given function, and so reduces memory size to store all the coefficients [15,22,24].

Example 2: Fig. 2 shows uniform and non-uniform segmentations of $\arcsin(X)$, where X has 6-bit accuracy, the function is approximated by quadratic polynomials, and the acceptable approximation error is 2^{-8}. The number of uniform segments is 32, while the number of non-uniform segments is only 4. (End of Example)

Input:	Numerical function $f(X)$, domain $[A,B)$ for X, accuracy m_{in} of X, polynomial order d, and acceptable approximation error ε_a.
Output:	Segments $[A,P_0),[P_0,P_1),\dots,[P_{t-2},B)$.

```
Step:
 1.  For [A,B), compute the maximum approximation error ε_d(A,B).
 2.  If ε_d(A,B) < ε_a or B − A ≤ 2^{−m_in}, then stop.
 3.  Else, partition [A,B) into two segments [A,P) and [P,B), where P = (A+B)/2.
 4.  Repeat Steps 1, 2, and 3 for each new segment recursively, until
     the maximum approximation errors are smaller than ε_a in all segments.
```

Fig. 3 Recursive segmentation algorithm for the domain.

3.2 Recursive Segmentation Algorithm

Although non-uniform segmentation yields fewer segments than uniform segmentation, non-uniform segmentation requires an additional circuit that maps a given X into a segment. Lee et al. [11] have proposed a special non-uniform segmentation, *hierarchical segmentation*, to simplify the additional circuit. However, since their method has only four segmentation types which simplify the additional circuit, the generated segmentation does not always adapt to the degree of nonlinearity of the given function. In this section, to reduce both hardware complexity and the number of segments, we present a new non-uniform segmentation method, *recursive segmentation*, that is a hybrid of the method [11] and our previous method [15,22,24].

Fig. 3 shows the recursive segmentation algorithm. The inputs for this algorithm are a numerical function $f(X)$, a domain $[A,B)$ for X, an accuracy m_{in} of X, a polynomial order d, and an acceptable approximation error ε_a. Then, this algorithm produces t segments $[A,P_0),[P_0,P_1),\dots,[P_{t-2},B)$ by recursively partitioning a segment into two equal-sized segments until achieving the acceptable approximation error ε_a in all segments. Note that this algorithm restricts the width w_i of each segment to $w_i = 2^{h_i} \times 2^{-m_{in}}$, where h_i is an integer. That is, the segmentation points P_i are restricted to values of which the least significant h_i bits are 0 (i.e., $P_i = (\dots\ p_{-j+1}\ p_{-j}\ 00\ \dots\ 0)_2$, where $j = m_{in} - h_i$). As shown in Fig. 3, the number of segments depends on the maximum approximation error $\varepsilon_d(A,B)$. In this paper, we use the Chebyshev approximation polynomials. For a segment $[S,E]$ of $f(X)$, the maximum approximation error of the dth-order Chebyshev approximation $\varepsilon_d(S,E)$ is given by [12]:

$$\varepsilon_d(S,E) = \frac{2(E-S)^{d+1}}{4^{d+1}(d+1)!} \max_{S \le X \le E} |f^{(d+1)}(X)|,$$

where $f^{(d+1)}$ is the $(d+1)$th-order derivative of f.

This algorithm can be applied to any given domain $[A,B)$. However, a wide domain necessarily requires a large number of segments. In this case, we can reduce the given domain to a narrower domain by using a range reduction technique [1, 13], as is done with existing methods based on uniform segmentation.

3.3 Computation of the Approximate Value

For each segment, $f(X)$ is approximated by the corresponding polynomial function $g(X,i)$. That is, the approximated value of $f(X)$ is computed by $g(X,i) = C_d(i)X^d + C_{d-1}(i)X^{d-1} + \dots + C_0(i)$, where i is a segment index assigned to each segment, and the coefficients $C_d(i), C_{d-1}(i), \dots, C_0(i)$ are derived from the dth-order Chebyshev approximation polynomial [12].

For each segment $[S_i, E_i)$, since $S_i \le X < E_i$ holds, we can offset X by S_i to compute the polynomial $g(X,i)$. By using the offset input $(X - S_i)$ instead of X, we reduce the size of multipliers needed to compute $g(X,i)$. By substituting $X - S_i + S_i$ for X, we transform $g(X,i)$ as follows:

$$\begin{aligned} g(X,i) &= C_d(i)X^d + C_{d-1}(i)X^{d-1} + \dots + C_0(i) \\ &= C_d(i)(X - S_i + S_i)^d \\ &\quad + C_{d-1}(i)(X - S_i + S_i)^{d-1} + \dots + C_0(i) \\ &= C_d(i)(X - S_i)^d \\ &\quad + \{C_{d-1}(i) + dC_d(i)S_i\}(X - S_i)^{d-1} + \dots \\ &\quad \dots + C_0(i). \end{aligned}$$

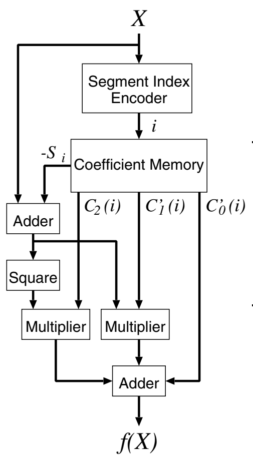
Let the coefficients of $(X - S_i)^j$ $(j = 0, 1, \dots, d-1)$ be

$$C'_j(i) = \sum_{k=0}^{d-j} \binom{j+k}{j} C_{j+k}(i) S_i^k \quad (j = 0, 1, \dots, d-1).$$

Then, we have

$$\begin{aligned} g(X,i) &= C_d(i)(X - S_i)^d + C'_{d-1}(i)(X - S_i)^{d-1} + \dots \\ &\quad \dots + C'_0(i). \end{aligned} \tag{1}$$

This transformation reduces the multiplier size (see Section 4.3). That is, instead of using the entire value X, as in the approximation $C_d(i)X^d + C_{d-1}(i)X^{d-1} + \dots + C_0(i)$ requiring the maximum number of bits to represent X, we use (1) to approximate the function, where typically smaller number of bits is needed to realize $X - S_i$.



$$\begin{array}{rl} X & =(x_{l-1}\, x_{l-2} \dots x_{-j}\, x_{-j-1} \dots x_{-m_{in}})_2 \\ S_i & =(s_{l-1}\, s_{l-2} \dots s_{-j}\ \ 0\ \ \dots\ \ 0\ )_2 \\ \hline X - S_i & =(\ 0\ \ \ 0\ \ \dots\ \ 0\ \ x_{-j-1} \dots x_{-m_{in}})_2 \end{array}$$

$$\Downarrow$$

$$\begin{array}{rl} X & =(x_{l-1}\, x_{l-2} \dots x_{-j}\, x_{-j-1} \dots x_{-m_{in}})_2 \\ \overline{S_i} & =(\overline{s_{l-1}}\, \overline{s_{l-2}} \dots \overline{s_{-j}}\ \ 1\ \ \dots\ \ 1\ )_2 \\ \hline X \& \overline{S_i} & =(\ 0\ \ \ 0\ \ \dots\ \ 0\ \ x_{-j-1} \dots x_{-m_{in}})_2 \end{array}$$

Note: 1. $x_{l-1} = s_{l-1}$, $x_{l-2} = s_{l-2}$, $\dots$, $x_{-j} = s_{-j}$, 2. $j = m_{in} - h_i$, and 3. '&' is bit-by-bit AND.

(a) Architecture of NFG. (b) Computation of $X - S_i$ using AND gates.

Fig. 4 Architecture of the NFG based on 2nd-order polynomials.

4. Architecture of the NFG

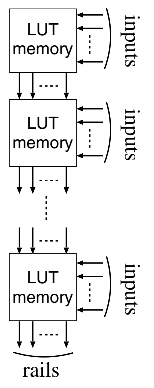
Fig. 4 shows the architecture of the NFG based on a 2nd-order polynomial. As shown in Fig. 4(a), polynomials of the form (1) are realized using a *segment index encoder* (*SIE*), a coefficient memory, circuits for $(X - S_i)^k$ $(k = d, d-1, \dots, 2)$, multipliers, and adders. Since modern FPGAs have logic elements, synchronous memory blocks, and dedicated multipliers, this architecture is efficiently implemented by those hardware resources in an FPGA. This architecture can realize any non-uniform segmentation. However, when recursive segmentation is used, we can realize $X - S_i$ using 2-input AND gates instead of an adder. As mentioned in the previous section, the least significant h_i bits of S_i are 0, and $X - S_i < 2^{h_i} \times 2^{-m_{in}}$ (i.e. $x_{l-1} = s_{l-1}, x_{l-2} = s_{l-2}, \dots, x_{-j} = s_{-j}$ and $s_{-j-1} = s_{-j-2} = \dots = s_{-m_{in}} = 0$ because of the way S_i is chosen.) Therefore, $X - S_i$ has 1's only in the least significant h_i bits, and these 1's occur in exactly the same position as the 1's in X. Thus, as shown in Fig. 4(b), we realize $X - S_i$ using AND gates driven on one side by $\overline{S_i}$, the complement of S_i. The SIE converts X into a segment index i. It realizes the segment index function $seg\_func(X) : \{0,1\}^n \to \{0, 1, \dots, t-1\}$ shown in Fig. 5(a), where X has n bits, and t denotes the number of segments.

4.1 Architecture of the SIE

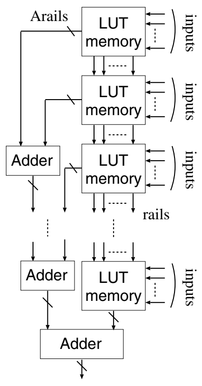
Fig. 5(b) shows an LUT cascade [22, 23] that realizes $seg\_func(X)$. The LUT cascade is obtained by functional decomposition using an MTBDD for $seg\_func(X)$ [20, 21], and can realize any $seg\_func(X)$, where the size of the LUT cascade depends on the number of segments. [15] has shown that this size can be reduced by reducing the number of segments. This section presents a new programmable architecture for the SIE that reduces the size and delay time. Fig. 5(c) shows the new architecture. To realize $seg\_func(X)$ using the SIE in Fig. 5(c), we represent

Segments	Index
$A \leq X < P_0$	0
$P_0 \leq X < P_1$	1
$\vdots$	$\vdots$
$P_{t-2} \leq X < B$	$t-1$

(a) Segment index function.

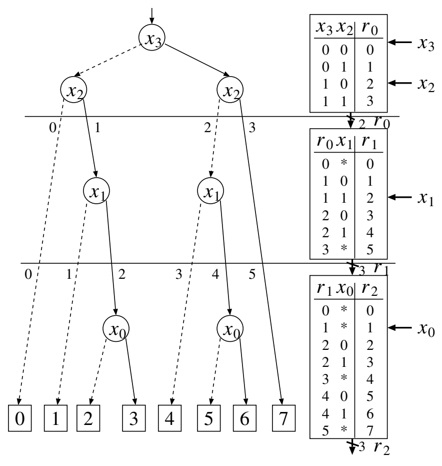


(b) LUT cascade (MT_SIE).



(c) LUT cascade and adders (EV_SIE).

Fig. 5 Segment index encoders.

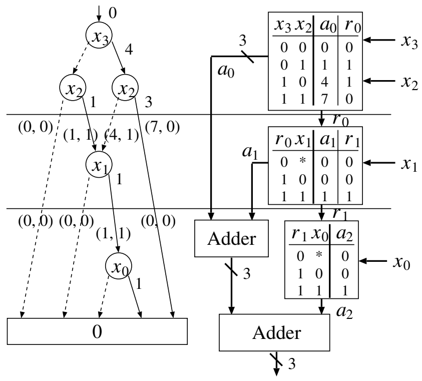


$x_3\,x_2$	r_0
0 0	0
0 1	1
1 0	2
1 1	3

$r_0\,x_1$	r_1
0 *	0
1 0	1
1 1	2
2 0	3
2 1	4
3 *	5

$r_1\,x_0$	r_2
0 *	0
1 *	1
2 0	2
2 1	3
3 *	4
4 0	5
4 1	6
5 *	7

(a) SIE using MTBDD (MT_SIE).



$x_3\,x_2$	a_0	r_0
0 0	0	0
0 1	1	1
1 0	4	1
1 1	7	0

$r_0\,x_1$	a_1	r_1
0 *	0	0
1 0	0	0
1 1	1	1

$r_1\,x_0$	a_2
0 *	0
1 0	0
1 1	1

(b) SIE using EVBDD (EV_SIE).

Fig. 6 Example of SIEs.

$seg\_func(X)$ using an EVBDD. And then, by decomposing the EVBDD, we obtain the SIE that consists of an LUT cascade and adders. In an LUT cascade, the interconnecting lines between adjacent LUT memories are called *rails*. In this case, the rails represent sub-functions in the EVBDD. The outputs from each LUT memory other than rails represent the sum of weights of edges. In this paper, we call such outputs *Arails (adder rails)*. To the best of our knowledge, this is the first design method using an EVBDD to produce the cascaded programmable architecture.

Example 3: By decomposing the MTBDD and EVBDD in Fig. 1, we obtain the SIEs in Fig. 6. Fig. 6(a) and (b) illustrate the correspondences between each LUT memory and decompositions of the MTBDD and the EVBDD, respectively. In these figures, the column labeled as 'r_i' in the table of each LUT denotes the rails that represent sub-functions in BDDs. The column 'a_i' in Fig. 6(b) denotes the Arails that represent the sum of weights of edges. In the MTBDD, numbers assigned to edges that cut across the horizontal lines represent sub-functions. In the EVBDD, "(a_i, r_i)" assigned to edges that cut across the horizontal lines represent the sum of weights and sub-functions, respectively. The SIE in Fig. 6(a) requires $2^2 \times 2 + 2^3 \times 3 + 2^4 \times 3 = 80$ bits and 3 levels (3 LUT memories). On the other hand, the SIE in Fig. 6(b) requires $2^2 \times 4 + 2^2 \times 2 + 2^2 \times 1 = 28$ bits and 4 levels (3 LUT memories + 1 adder). (End of Example)

This paper uses two terms: *MT_SIE* and *EV_SIE* denote the SIEs designed using an MTBDD (Fig. 5(b)) and an EVBDD (Fig. 5(c)), respectively. Both the MT_SIE and the EV_SIE can realize any non-uniform segmentation. In both cases, the size of LUT memories depends on the number of segments. Specifically,

Theorem 1: Let $seg\_func(X)$ be a segment index function with t segments. Then, there exists an EV_SIE for $seg\_func(X)$ with at most $\lceil \log_2 t \rceil$ rails and $\lceil \log_2 t \rceil$ Arails.

Proof: See Appendix.

The size of LUT memories and the number of levels of an EV_SIE depend on the decomposition of an EVBDD. To obtain the optimum decomposition, we can use optimization algorithms for heterogeneous multi-valued decision diagrams (MDDs) [14].

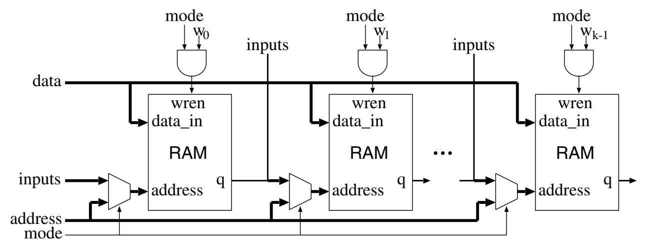


Fig. 7 LUT cascade implemented by embedded RAMs.

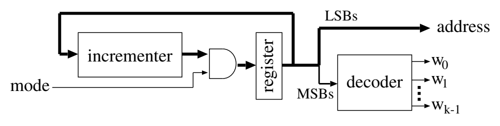


Fig. 8 Control circuit for the SIE.

4.2 Programmability of the SIE

The LUT memories of the SIEs shown in Fig. 5(b) and (c) are implemented by embedded RAMs (e.g. M4Ks) in an FPGA. Thus, by changing the data for the LUT memories and the coefficient memory, a wide class of numerical functions can be realized by a single architecture. Since just changing the RAM data can switch numerical functions, we can switch functions even while the FPGA is running.

Fig. 7 and Fig. 8 show the details of the LUT cascade and the control circuit for changing the RAM data, respectively. In these figures, 'mode' denotes a signal to switch between the operation mode and the program mode of the LUT cascade. The control circuit consists of a counter and a decoder, and generates address and write enable signal for each RAM sequentially.

To the best of our knowledge, a programmable architecture for the SIE has never before been proposed.

4.3 Reduction of the Size of the Multiplier

Since large multipliers have large delay, it is important to reduce multiplier size. We do this in two ways; Reduce the number of bits needed to represent 1. the coefficients and 2. the variables $(X - S_i)$.

To reduce the number of bits in the coefficients, we use a *scaling method* [10]. We first shift right the coefficients. Then, we apply rounding. Then, we do the actual multiplication. And, finally, we shift left the product to compensate for the original shift right of the coefficients. This process is similar to floating point multiplication. A side effect is that rounding error is increased, since rounding occurs on a smaller value. In applying this method, we choose the largest exponent (right shift) that produces an error no greater than the given acceptable error [15]. If this yields an exponent of 0 (no right shift), in all segments, then we do not use the scaling method.

To reduce the value of the variable $X - S_i$, we make the following observation. In each segment $[S_i, E_i)$, we have $X - S_i < E_i - S_i$. Thus, reducing the segment width $E_i - S_i$ reduces $X - S_i$ for X near E_i. However, this also increases the number of segments, and thus the size of coefficient memory. We show a segment reduction technique that does not increase the coefficient memory size.

In an FPGA implementation, the coefficient memory in Fig. 4 has 2^u words, where $u = \lceil \log_2 t \rceil$ and t is the number of segments. Therefore, we can increase the number of segments up to $t = 2^u$ without increasing the coefficient memory size. From Theorem 1, the size of the EV_SIE also depends on the value of u. Increasing the number of segments to $t = 2^u$ rarely increases the size of the EV_SIE. We reduce the size of segments by dividing the largest segment into two equal sized segments up to $t = 2^u$.

5. Experimental Results

5.1 Number of Segments and Computation Time

Table 1 compares the number of segments for various segmentation methods based on a 2nd-order Chebyshev approximation. In Table 1, "No. of uniform segs" shows the number of uniform segments, "No. of nonuni. segs" shows the number of non-uniform segments produced by [15], and "Recursive" denotes the recursive segmentation method shown in this paper. In the column "Recursive", the sub-column "No. of segs 1" shows the number of segments produced by the segmentation algorithm shown in Section 3. The sub-column "No. of segs 2" shows the number of segments produced by additionally applying the reduction method of multiplier size shown in Section 4. The sub-column "Time" shows the total CPU time, in milliseconds, for both the segmentation algorithm and the reduction method of multiplier size.

Table 1 Number of segments for various segmentation methods.

X has 23-bit accuracy.
Acceptable approximation error: 2^{-25}

Function	Domain	No. of	No. of	Recursive		
$f(X)$	$[A,B)$	uniform	nonuni.	No. of	No. of	Time
		segs	segs	segs 1	segs 2	[msec.]
e^X	$[0,1)$	128	67	103	128*	10
$\sin(\pi X)$	$[0,0.5)$	128	74	112	128*	10
$\tan(\pi X)$	$[0,0.5)$	4,194,304	4,594	5,723	8,192	1,600
$\arcsin(X)$	$[0,1)$	8,388,608	256	363	512	70
$\sqrt{X}$	$(0,1)$	8,388,607	228	322	512	30
$\sqrt{-\ln(X)}$	$(0,1)$	8,388,607	698	967	1,024	190
$X\ln(X)$	$(0,1)$	2,097,152	172	250	256	10

*Uniform segmentation is produced.
Environment: Sub Blade 2500 (Silver), UltraSPARC-IIIi 1.6GHz, 6GB memory, Solaris 9.

Table 1 shows that uniform segmentation requires excessively many segments to approximate certain functions, such as $\tan(\pi X)$. Existing methods based on uniform segmentation cannot implement those functions in conventional FPGAs because the required coefficient memory is too large. Actually, many existing methods have not realized $\tan(\pi X)$ in domain $[0, 0.5)$. This is because $\tan(\pi X)$ in $[0, 0.5)$ can be computed by $\sin(\pi X)/\cos(\pi X)$ or a combination of $\tan(\pi X)$ in $[0, 0.25]$ and $1/\tan(\pi X')$, where $X' = 0.5 - X$, and those functions can be implemented by the existing methods. However, these require multiple NFGs that realize elementary functions, such as sin, cos, or the reciprocal function. On the other hand, the non-uniform or recursive segmentation described here can compactly realize $\tan(\pi X)$ with a single NFG, since non-uniform and recursive segmentation methods require many fewer segments. For all functions in Table 1, the non-uniform segmentation method [15] requires the fewest segments among the three segmentation methods. Although our recursive segmentation algorithm restricts the segmentation points, it requires only up to 2.2 times more segments than non-uniform segmentation [15]. That is, our recursive segmentation algorithm generates a segmentation appropriate to the given function, while restricting the segmentation points. Thus, our recursive segmentation produces a small coefficient memory for the given function. In the next section, we show that recursive segmentation reduces the size of SIE, as well.

Table 2 FPGA implementation of SIEs.

FPGA device:	Altera Stratix EP1S10F484C5 (LE: 10,570, M4K: 60, M512: 90)
Logic synthesis tool:	Altera QuartusII 5.0 (speed optimization, timing requirement of 200MHz)

Function $f(X)$	Optimum non-uniform segmentation								Recursive segmentation							
	MT_SIE				EV_SIE				MT_SIE				EV_SIE			
	LUT size [bits]	LE	Level	Delay [nsec.]	LUT size [bits]	LE	Level	Delay [nsec.]	LUT size [bits]	LE	Level	Delay [nsec.]	LUT size [bits]	LE	Level	Delay [nsec.]
e^X	26,368	115	8	58.8	23,040	154	7	43.5	0	0	0	0	0	0	0	0
$\sin(\pi X)$	26,880	115	8	54.9	23,552	151	7	45.1	0	0	0	0	0	0	0	0
$\tan(\pi X)$	1,802,240	–	5	–	179,968	340	10	75.3	1,687,552	–	5	–	15,108	201	7	43.6
$\arcsin(X)$	61,440	123	8	53.2	53,824	336	13	87.6	49,152	109	7	46.7	9,984	107	5	28.0
$\sqrt{X}$	61,440	123	8	53.2	57,408	289	11	75.5	44,544	107	7	46.4	10,752	112	5	27.2
$\sqrt{-\ln(X)}$	266,240	138	7	56.7	116,160	330	11	81.4	172,032	118	7	54.3	12,736	148	6	38.4
$X\ln(X)$	61,440	129	8	52.5	48,400	250	10	63.8	20,992	67	5	30.8	8,960	74	4	22.9

–: It cannot be mapped into the FPGA due to insufficient RAM blocks.

Table 3 FPGA implementation of 23-bit precision (23-bit accuracy) NFGs.

FPGA device:	Altera Stratix EP1S60F1020C5
	(LE: 57,120, DSP: 144, M4K: 292, M512: 574)
Logic synthesis tool:	Altera QuartusII 5.0
	(speed optimization, timing requirement of 200MHz)

Function $f(X)$	MTNFG based on optimum nonuni.					EVNFG based on recursive				
	Memory [bits]	LE	DSP	Level	Delay [nsec.]	Memory [bits]	LE	DSP	Level	Delay [nsec.]
e^X	39,040	689	10	13	99.6	8,064	432	10	3	25.1
$\sin(\pi X)$	36,864	635	10	13	99.1	7,936	395	10	3	28.3
$\tan(\pi X)$	2,867,200	–	16	11	–	973,572	1,059	16	12	92.3
$\arcsin(X)$	84,736	1,301	16	14	107.3	53,504	937	16	10	80.3
$\sqrt{X}$	83,712	1,041	16	14	116.5	53,760	917	16	10	77.2
$\sqrt{-\ln(X)}$	357,376	950	16	13	99.8	103,872	972	16	11	88.3
$X\ln(X)$	83,200	988	16	14	116.0	31,744	989	16	9	70.4

–: It cannot be mapped into the FPGA due to insufficient RAM blocks.

5.2 FPGA Implementation of SIEs

Table 2 compares the FPGA implementation results of the MT_SIE and EV_SIE. In this table, "LUT size" shows the total size of LUT memories used in the SIE, in bits. Note that the size of LUT memory and LE, the number of logic elements, are 0 for e^X and $\sin(\pi X)$ when recursive segmentation is used. This is because our algorithm generated uniform segments as shown in Table 1, and so an SIE was not needed. In the experiment that produced the data in Table 2, we optimized the decomposition of the MTBDDs and EVBDDs by requiring the size of each LUT memory used in these SIEs to be 4K bits, the same as the RAM block (M4K) of the FPGA.

Table 2 shows that, for optimum non-uniform segmentation, the EV_SIEs have smaller LUT memory size than the MT_SIEs. For example, for $\tan(\pi X)$, the LUT memory size of the EV_SIE is only 10% of the size needed by the MT_SIE. For $\tan(\pi X)$, the LUT memory size of MT_SIE is quite large because the number of non-uniform segments is large. From experiments with uniform segmentation, we know that for $\tan(\pi X)$, the total memory size needed by the NFG using the MT_SIE is only 1.5% of the total memory size needed by the NFG based on uniform segmentation (i.e. the existing methods). However, this is still too large to implement with the FPGA. By using the EV_SIE, we can reduce the LUT memory size significantly, and make the NFG implementable with the FPGA.

Our recursive segmentation can reduce both the LUT memory size and the delay time of the MT_SIEs. Especially, for $X\ln(X)$, using an MT_SIE designed for recursive segmentation has only 34% of the LUT memory size and 59% of the delay of the MT_SIE designed for optimum non-uniform segmentation.

By using recursive segmentation and the EV_SIE, we can reduce both LUT memory size and delay time of SIEs significantly. For all functions in Table 2, both LUT memory size and delay time of the EV_SIEs for recursive segmentation are much smaller for the MT_SIEs. In terms of the number of LEs, the EV_SIEs for recursive segmentation require only up to 1.3 times more LEs than the MT_SIEs. Therefore, designing an EV_SIE for recursive segmentation yields faster and more compact SIEs than obtained by previous methods. The design is formal and is easily programmed.

5.3 FPGA Implementation of NFGs

Table 3 compares the FPGA implementation results of our NFGs using EV_SIE (EVNFGs) with the existing NFGs using MT_SIE (MTNFGs) [15], where EVNFGs are based on recursive segmentation and MTNFGs are based on the optimum non-uniform segmentation. Both NFGs have 23-bit precision (23-bit accuracy).

Table 4 FPGA implementation of 24-bit precision NFGs.

FPGA device:	Xilinx Virtex-II XC2V4000-6
Logic synthesis tool:	Synplify Premier Ver. 8.5

Function	NFG in [11]					EVNFG				
$f(X)$	Memory [bits]	Slice	Mult.	Level	Delay [nsec.]	Memory [bits]	Slice	Mult.	Level	Delay [nsec.]
$X\ln(X)$	40,446	871	10	14	103.7	34,560	454	5	9	64.7
humps	NA	409	4	13	82.8	91,648	189	4	8	55.9

Table 5 Comparison of design methods.

	Segmentation methods	
SIEs	Non-uniform	Recursive
MT	· **Smaller** coefficient memory · **Largest** and **slower** SIE	· **Larger** coefficient memory · **Smaller** and **faster** SIE
EV	· **Smaller** coefficient memory · **Larger** and **slowest** SIE	· **Larger** coefficient memory · **Smallest** and **fastest** SIE

From Table 2 and Table 3, we can see that the LUT memory size of MT_SIE accounts for more than 2/3 of the total memory size of the MTNFG. On the other hand, by using recursive segmentation and EV_SIE, the LUT memory size needed for the SIE can be reduced to less than 1/4 of the total memory size of the EVNFG. Thereby, the EVNFGs require only 21% to 64% of memory size needed for the MTNFGs. For $\arcsin(X)$ and $\sqrt{X}$, as shown in Table 1, our recursive segmentation requires a coefficient memory that is about twice as large as needed for the optimum non-uniform segmentation. Nevertheless, by using EV_SIEs, the total memory sizes of EVNFGs can be reduced to about 60% of the memory sizes of MTNFGs. Further, Table 3 shows that EVNFGs require fewer LEs and levels (i.e., shorter latency) than MTNFGs, and the delay time of EVNFGs is only about 24% to 94% of the delay time of MTNFGs.

To compare our NFG with the existing NFG based on another non-uniform segmentation method (hierarchical segmentation) shown in [11], we implemented our 24-bit precision NFGs for $X\ln(X)$ and “humps” function using the Xilinx Virtex-II FPGA (XC2V4000-6) and the Synplify Premier 8.5. The humps function is a quotient of polynomials

$$\text{humps} = \frac{0.0004x + 0.0002}{x^4 - 1.96x^3 + 1.348x^2 - 0.378x + 0.0373}.$$

Table 4 compares the FPGA implementation results of our NFGs and the NFGs shown in [11]. For the humps function, the memory size of the NFG is not shown in [11].

From these results, we can see that our NFGs using recursive segmentation and the EV_SIE can realize a wide range of functions faster and more compactly than existing NFGs.

5.4 Comparison of Design Methods

As for the segmentation, we have two methods: optimum non-uniform and recursive, and as for the SIE, we have two methods: MT_SIE and EV_SIE. Thus, there exist four different design methods. Table 5 compares four design methods: they are abbreviated as Non-uniform_MT, Recursive_MT, Non-uniform_EV, and Recursive_EV.

Roughly speaking, the non-uniform segmentation produces a smaller coefficient memory, but a larger and slower SIE. On the other hand, the recursive segmentation produces a larger coefficient memory, but a smaller and faster SIE. As for the SIE, the EV_SIE requires smaller LUT memories than the MT_SIE. However, the EV_SIE requires a cascade of adders that often makes the NFG slower than one with the MT_SIE.

Non-uniform_MT produces a small coefficient memory, but its SIE has the largest LUT memory size among the four methods. Thus, Non-uniform_MT results in NFGs with large memory size.

Recursive_MT produces a smaller and faster SIE than Non-uniform_MT. However, the reduction of LUT memory size in the SIE is insufficient to compensate for the increase in the coefficient memory. Thus, NFGs with Recursive_MT are faster than with Non-uniform_MT, but still require large memory size.

Non-uniform_EV produces a small coefficient memory and the SIE with smaller LUT memories than Non-uniform_MT. However, the SIE is the slowest among the four methods. Thus, NFGs with Non-uniform_EV require smaller memory size than with Non-uniform_MT, but they are the slowest among the four methods.

Recursive_EV produces the smallest and the fastest SIE among the four. And, the reduction of LUT memory size in the SIE is sufficient to compensate for the increase in the coefficient memory. Thus, NFGs with Recursive_EV are the smallest and the fastest among the four.

6. Concluding Remarks

We have presented design methods for numerical function generators using recursive segmentation and EVBDDs. Our recursive segmentation is a hybrid approach of an optimum non-uniform segmentation that produces the fewest segments and a segmentation that reduces hardware complexity. Thus, our recursive segmentation reduces the sizes of both the coefficient memory and the SIE. We have proposed a new programmable architecture and its design method using EVBDDs. We have shown that using both the new segmentation method and the new architecture can produce faster and more compact programmable NFGs than the existing NFGs. We also show that an adder can be replaced by a set of 2-input AND gates, thus reducing the delay. Experimental results show

1. recursive segmentation yields MT_SIEs that have only 49% of the LUT memory size and 55% of the delay of MT_SIEs based on optimum segmentation
2. by using EVBDDs to realize recursive segmentation, we reduce LUT memory size and delay of the segment index encoders. On the average, this yields EV_SIEs that require only 8% of the LUT memory size and 36% of the delay of segment index encoders designed by optimum non-uniform segmentation, and
3. overall, our NFGs require, on the average, only 39% of the memory and 58% of the delay associated with NFGs based on MT_SIEs and optimum non-uniform segmentation.

These results are for a suite of seven functions, including e^x, $\sin(\pi x)$, and $\sqrt{x}$.

Acknowledgments

This research is partly supported by the Grant in Aid for Scientific Research of the Japan Society for the Promotion of Science (JSPS), funds from Ministry of Education, Culture, Sports, Science, and Technology (MEXT) via Kitakyushu innovative cluster project, a contract with the National Security Agency, the MEXT Grant-in-Aid for Young Scientists (B), 18700048, 2007, and Hiroshima City University Grant for Special Academic Research (General Studies), 6101, 2007. The comments of two reviewers were useful in improving this paper. Discussion with Prof. Yukihiro Iguchi improved this paper.

References

[1] N. Brisebarre, D. Defour, P. Kornerup, J.-M. Muller, and N. Revol, "A new range-reduction algorithm," *IEEE Trans. Comput.*, Vol. 54, No. 3, pp. 331–339, Mar. 2005.

[2] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, Vol. C-35, No. 8, pp. 677–691, Aug. 1986.

[3] A. Cantoni, "Optimal curve fitting with piecewise linear functions," *IEEE Trans. on Comp.*, Vol. 20, No. 1, pp. 59–67, Jan. 1971.

[4] J. Cao, B. W. Y. Wei, and J. Cheng, "High-performance architectures for elementary function generation," *Proc. of the 15th IEEE Symp. on Computer Arithmetic (ARITH'01)*, Vail, Colorado, pp. 136–144, June 2001.

[5] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang, "Spectral transforms for large Boolean functions with applications to technology mapping," *Proc. of 30th ACM/IEEE Design Automation Conference*, pp. 54–60, June 1993.

[6] D. Defour, F. de Dinechin, and J.-M. Muller, "A new scheme for table-based evaluation of functions," *36th Asilomar Conference on Signals, Systems, and Computers,*, Pacific Grove, California, pp. 1608–1613, Nov. 2002.

[7] J. Detrey and F. de Dinechin, "Table-based polynomials for fast hardware function evaluation," *16th IEEE Inter. Conf. on Application-Specific Systems, Architectures, and Processors (ASAP'05)*, pp. 328–333, 2005.

[8] V. K. Jain, S. A. Wadekar, and L. Lin, "A universal nonlinear component and its application to WSI," *IEEE Trans. on Components, Hybrids, and Manufacturing Technology*, Vol. 16, No. 7, pp. 656–664, Nov. 1993.

[9] Y-T. Lai, M. Pedram, and S. B. Vrudhula, "EVBDD-based algorithms for linear integer programming, spectral transformation and functional decomposition," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, Vol. 13, No. 8, pp. 959–975, Aug. 1994.

[10] D.-U. Lee, W. Luk, J. Villasenor, and P. Y. K. Cheung, "Non-uniform segmentation for hardware function evaluation," *Proc. Inter. Conf. on Field Programmable Logic and Applications*, pp. 796–807, Lisbon, Portugal, Sept. 2003.

[11] D.-U. Lee, W. Luk, J. Villasenor, and P. Y. K. Cheung, "Hierarchical segmentation schemes for function evaluation," *Proc. of the IEEE Conf. on Field-Programmable Technology*, Tokyo, Japan, pp. 92–99, Dec. 2003.

[12] J. H. Mathews, *Numerical Methods for Computer Science, Engineering and Mathematics*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1987.

[13] J.-M. Muller, *Elementary Function: Algorithms and Implementation*, Birkhauser Boston, Inc., Secaucus, NJ, 1997.

[14] S. Nagayama and T. Sasao, "On the optimization of heterogeneous MDDs," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, Vol. 24, No. 11, pp. 1645–1659, Nov. 2005.

[15] S. Nagayama, T. Sasao, and J. T. Butler, "Compact numerical function generators based on quadratic approximation: architecture and synthesis method," *IEICE Trans. on Fundamentals*, Vol. E89-A, No. 12, pp. 3510–3518, Dec. 2006.

[16] S. Nagayama, T. Sasao, and J. T. Butler, "Numerical function generators using edge-valued binary decision diagrams," *Proc. of Asia and South Pacific Design Automation Conference (ASPDAC'07)*, Yokohama, Japan, pp. 535–540, 2007.

[17] S. Nagayama, T. Sasao, and J. T. Butler, "Design method for numerical function generators based on polynomial approximation for FPGA implementation," *Proc. of 10th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools (DSD'07)*, Germany, Aug. 2007 (accepted).

[18] J.-A. Piñeiro, S. F. Oberman, J.-M. Muller, and J. D. Bruguera, "High-speed function approximation using a minimax quadratic interpolator," *IEEE Trans. on Comp.*, Vol. 54, No. 3, pp. 304–318, Mar. 2005.

[19] T. Sasao and M. Fujita (eds.), *Representations of Discrete Functions*, Kluwer Academic Publishers 1996.

[20] T. Sasao, M. Matsuura, and Y. Iguchi, "A cascade realization of multiple-output function for reconfigurable hardware," *Inter. Workshop on Logic Synthesis (IWLS'01)*, Lake Tahoe, CA, pp. 225–230, June 12–15, 2001.

[21] T. Sasao and M. Matsuura, "A method to decompose multiple-output logic functions," *41st Design Automation Conference*, San Diego, CA, pp. 428–433, June 2–6, 2004.

[22] T. Sasao, J. T. Butler, and M. D. Riedel, "Application of LUT cascades to numerical function generators," *Proc. the 12th Workshop on Synthesis and System Integration of Mixed Information Technologies (SASIMI'04)*, Kanazawa, Japan, pp. 422–429, Oct. 2004.

[23] T. Sasao, S. Nagayama, and J. T. Butler, "Programmable numerical function generators: architectures and synthesis method," *Proc. Inter. Conf. on Field Programmable Logic and Applications (FPL'05)*, Tampere, Finland, pp. 118–123, Aug. 2005.

[24] T. Sasao, S. Nagayama, and J. T. Butler, "Numerical function generators using LUT cascades," *IEEE Transactions on Computers*, Vol. 56, No. 6, pp. 826–838, Jun. 2007.

[25] M. J. Schulte and E. E. Swartzlarnder, "Hardware designs for exactly rounded elementary functions," *IEEE Trans. on Comp.*, Vol. 43, No. 8, pp. 964–973, Aug. 1994.

[26] J. E. Stine and M. J. Schulte, "The symmetric table addition method for accurate function approximation," *Jour. of VLSI Signal Processing*, Vol. 21, No. 2, pp. 167–177, June 1999.

[27] M. R. Williams, *History of Computing Technology*, IEEE Computer Society Press, Los Alamitos, CA, 1997.

Appendix A: Proof of Theorem 1

Theorem 1: Let $seg\_func(X)$ be a segment index function with t segments. Then, there exists an EV_SIE for $seg\_func(X)$ with at most $\lceil \log_2 t \rceil$ rails and $\lceil \log_2 t \rceil$ Arails.

Proof: The second part of the hypothesis concerning the number of Arails was proven in [22]. Specifically, it was shown that

Theorem A: [22] Let $seg\_func(X)$ be a segment index function with t segments. Then, there exists an LUT cascade for $seg\_func(X)$ with at most $\lceil \log_2 t \rceil$ rails.

Each node in an MTBDD represents the Shannon expansion: $x_i \cdot f_1 + \bar{x}_i \cdot f_0$, where f_0 and f_1 are sub-functions with respect to $x_i = 0$ and $x_i = 1$, respectively. On the other hand, each node in an EVBDD represents the expansion:

$$x_i(\alpha + f'_1) + \bar{x}_i \cdot f_0,$$

where $f_1 = \alpha + f'_1$, and α is a constant value of the sub-function f_1. Note that, in an EVBDD, α is the weight of a 1-edge. Let μ_e be the number of distinct sub-functions produced by this expansion, and let μ_s be the number of distinct sub-functions by the Shannon expansion. Then, we have

$$\mu_e \leq \mu_s.$$

Theorem A shows that $seg\_func(X)$ can be represented by an MTBDD in which the number of distinct sub-functions with respect to each x_i is at most t. Thus, there exists an EVBDD for $seg\_func(X)$ that has at most t distinct sub-functions with respect to each x_i. In an EVBDD for $seg\_func(X)$, the sum of weights of edges on a path is at most t, since each weight is a non-negative integer. Therefore, we have Theorem 1. ∎

Shinobu NAGAYAMA received the B.S. and M.E. degrees from the Meiji University, Kanagawa, Japan, in 2000 and 2002, respectively, and the Ph.D. degree in computer science from the Kyushu Institute of Technology, Iizuka, Japan, in 2004. He is now a research associate at the Hiroshima City University, Hiroshima, Japan. He received the Outstanding Contribution Paper Award from the IEEE Computer Society Technical Committee on Multiple-Valued Logic (MVL-TC) in 2005 for a paper presented at the International Symposium on Multiple-Valued Logic in 2004, and the Excellent Paper Award from the Information Processing Society of Japan (IPS) in 2006. His research interest includes numerical function generators, decision diagrams, software synthesis, and embedded systems.

Tsutomu SASAO received the BE, ME, and PhD degrees in electronics engineering from Osaka University, Osaka, Japan, in 1972, 1974, and 1977, respectively. He has held faculty/research positions at Osaka University, Japan, the IBM T.J. Watson Research Center, Yorktown Heights, New York, and the Naval Postgraduate School, Monterey, California. He is now a Professor of the Department of Computer Science and Electronics at the Kyushu Institute of Technology, Iizuka, Japan. His research areas include logic design and switching theory, representations of logic functions, and multiple-valued logic. He has published more than nine books on logic design, including *Logic Synthesis and Optimization, Representation of Discrete Functions, Switching Theory for Logic Synthesis*, and *Logic Synthesis and Verification*, Kluwer Academic Publishers, 1993, 1996, 1999, and 2001, respectively. He has served as Program Chairman for the IEEE International Symposium on Multiple-Valued Logic (ISMVL) many times. Also, he was the Symposium Chairman of the 28th ISMVL held in Fukuoka, Japan, in 1998. He received the NIWA Memorial Award in 1979, Distinctive Contribution Awards from the IEEE Computer Society MVL-TC for papers presented at ISMVLs in 1986, 1996, 2003 and 2004, and Takeda Techno-Entrepreneurship Award in 2001. He has served as an Associate Editor of the *IEEE Transactions on Computers*. He is a fellow of the IEEE.

Jon T. BUTLER received the BEE and MEngr degrees from Rensselaer Polytechnic Institute, Troy, New York, in 1966 and 1967, respectively. He received the PhD degree from The Ohio State University, Columbus, in 1973. Since 1987, he has been a professor at the Naval Postgraduate School, Monterey, California. From 1974 to 1987, he was at Northwestern University, Evanston, Illinois. During that time, he served two periods of leave at the Naval Postgraduate School, first as a National Research Council Senior Postdoctoral Associate (1980-1981) and second as the NAVALEX Chair Professor (1985-1987). He served one period of leave as a foreign visiting professor at the Kyushu Institute of Technology, Iizuka, Japan. His research interests include logic optimization and multiple-valued logic. He has served on the editorial boards of the *IEEE Transactions on Computers*, *Computer*, and IEEE Computer Society Press. He has served as the editor-in-chief of *Computer* and IEEE Computer Society Press. He received the Award of Excellence, the Outstanding Contributed Paper Award, and a Distinctive Contributed Paper Award for papers presented at the International Symposium on Multiple-Valued Logic. He received the Distinguished Service Award, two Meritorious Awards, and nine Certificates of Appreciation for service to the IEEE Computer Society. He is a fellow of the IEEE.